In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
df = pd.read_csv('data/credit_card_default.csv')
df.head(10)
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6
0	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	0
1	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	3261
2	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	15549
3	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	29547
4	5	50000	1	2	1	57	-1	0	-1	0	...	20940	19146	19131
5	6	50000	1	1	2	37	0	0	0	0	...	19394	19619	20024
6	7	500000	1	1	2	29	0	0	0	0	...	542653	483003	473944
7	8	100000	2	2	2	23	0	-1	-1	0	...	221	-159	567
8	9	140000	2	3	1	28	0	0	2	0	...	12211	11793	3719
9	10	20000	1	3	2	35	-2	-2	-2	-2	...	0	13007	13912

10 rows × 25 columns

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          30000 non-null  int64
 1   LIMIT_BAL                   30000 non-null  int64
 2   SEX                         30000 non-null  int64
 3   EDUCATION                   30000 non-null  int64
 4   MARRIAGE                    30000 non-null  int64
 5   AGE                         30000 non-null  int64
 6   PAY_1                       30000 non-null  int64
 7   PAY_2                       30000 non-null  int64
 8   PAY_3                       30000 non-null  int64
 9   PAY_4                       30000 non-null  int64
 10  PAY_5                       30000 non-null  int64
 11  PAY_6                       30000 non-null  int64
 12  BILL_AMT1                   30000 non-null  int64
 13  BILL_AMT2                   30000 non-null  int64
 14  BILL_AMT3                   30000 non-null  int64
 15  BILL_AMT4                   30000 non-null  int64
 16  BILL_AMT5                   30000 non-null  int64
 17  BILL_AMT6                   30000 non-null  int64
 18  PAY_AMT1                    30000 non-null  int64
 19  PAY_AMT2                    30000 non-null  int64
 20  PAY_AMT3                    30000 non-null  int64
 21  PAY_AMT4                    30000 non-null  int64
 22  PAY_AMT5                    30000 non-null  int64
 23  PAY_AMT6                    30000 non-null  int64
 24  default payment next month  30000 non-null  int64
dtypes: int64(25)
memory usage: 5.7 MB
```

In [ ]:

```
df.rename(columns = lambda x: x.lower(), inplace=True)
df.rename(columns = {"default payment next month":"default"}, inplace=True)
df.columns
```

Out[ ]:

```
Index(['id', 'limit_bal', 'sex', 'education', 'marriage', 'age', 'pay_1',
       'pay_2', 'pay_3', 'pay_4', 'pay_5', 'pay_6', 'bill_amt1', 'bill_amt2',
       'bill_amt3', 'bill_amt4', 'bill_amt5', 'bill_amt6', 'pay_amt1',
       'pay_amt2', 'pay_amt3', 'pay_amt4', 'pay_amt5', 'pay_amt6', 'default'],
      dtype='object')
```

In [ ]:

```
df.nunique()
```

Out[ ]:

```
id           30000
limit_bal       81
sex              2
education        7
marriage         3
age             56
pay_1           11
pay_2           11
pay_3           11
pay_4           11
pay_5           10
pay_6           10
bill_amt1    22723
bill_amt2    22346
bill_amt3    22026
bill_amt4    21548
bill_amt5    21010
bill_amt6    20604
pay_amt1      7943
pay_amt2      7899
pay_amt3      7518
pay_amt4      6937
pay_amt5      6897
pay_amt6      6939
default          2
dtype: int64
```

In [ ]:

```
df["male"] = (df['sex'] == 1).astype(int)
df['grad_school'] = (df['education']==1).astype(int)
df["university"] = (df['education']==2).astype(int)
df["married"] = (df['marriage']==1).astype(int)
```

In [ ]:

```
bill_amt_features = ["bill_amt" + str(i) for i in range(1,7)]
pay_amt_features = ["pay_amt" + str(i) for i in range(1,7)]
```

In [ ]:

```
binary_features = ['male', 'grad_school', 'university', 'married']
pay_features = ["pay_" + str(i) for i in range(1,7)]
num_features = ["limit_bal", "age"] + bill_amt_features + pay_amt_features + pay_features
```

In [ ]:

```
x = df[num_features + binary_features]
y = df['default']
```

In [ ]:

```
x
```

Out[ ]:

	limit_bal	age	bill_amt1	bill_amt2	bill_amt3	bill_amt4	bill_amt5	bill_amt6	pay_amt1	pay_amt2	...	pay_1	pay_2	pay_
0	20000	24	3913	3102	689	0	0	0	0	689	...	2	2	-
1	120000	26	2682	1725	2682	3272	3455	3261	0	1000	...	-1	2	
2	90000	34	29239	14027	13559	14331	14948	15549	1518	1500	...	0	0	
3	50000	37	46990	48233	49291	28314	28959	29547	2000	2019	...	0	0	
4	50000	57	8617	5670	35835	20940	19146	19131	2000	36681	...	-1	0	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
29995	220000	39	188948	192815	208365	88004	31237	15980	8500	20000	...	0	0	
29996	150000	43	1683	1828	3502	8979	5190	0	1837	3526	...	-1	-1	-
29997	30000	37	3565	3356	2758	20878	20582	19357	0	0	...	4	3	
29998	80000	41	-1645	78379	76304	52774	11855	48944	85900	3409	...	1	-1	
29999	50000	46	47929	48905	49764	36535	32428	15313	2078	1800	...	0	0	

30000 rows × 24 columns

In [ ]:

```
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, RepeatedStratifiedKFolc
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score
```

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state = 134)
```

In [ ]:

```
sc_features = ["limit_bal", "age"] + bill_amt_features + pay_amt_features
```

In [ ]:

```
sc = StandardScaler()
```

In [ ]:

```
x_train.loc[:,sc_features] = sc.fit_transform(x_train[sc_features])
x_test.loc[:,sc_features] = sc.transform(x_test[sc_features])
```

In [ ]:

```
x_train.head()
```

Out[ ]:

	limit_bal	age	bill_amt1	bill_amt2	bill_amt3	bill_amt4	bill_amt5	bill_amt6	pay_amt1	pay_amt2	...	pay_1	pay
29245	-0.911109	-1.136689	-0.055124	-0.613938	-0.591388	-0.569915	-0.523784	-0.584140	-0.221162	-0.183388	...	0	
26498	-0.911109	1.790000	-0.039584	-0.001461	-0.127831	-0.672645	-0.490840	-0.652265	-0.219044	-0.177226	...	0	
20130	-0.678940	0.814437	0.238241	0.293119	0.358289	0.454690	0.548120	0.609451	-0.154595	-0.097121	...	2	
25126	4.428778	1.573208	3.676412	3.754222	3.891379	3.946858	4.047345	3.870115	0.444513	0.297247	...	0	
20337	-0.911109	2.765562	0.021160	0.029323	0.068152	0.112776	-0.165604	-0.143091	-0.341952	-0.148635	...	3	

5 rows × 24 columns

KNN Classifier

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:

```
# commented for being computationally intensive
'''param_grid_knn = {
    'n_neighbors': np.arange(5,15),
    'p': (1,2),
    'weights': ('uniform', 'distance'),
    'metric': ('minkowski', 'chebyshev')
}
knn_gscv = KNeighborsClassifier()
knn_gscv = GridSearchCV(knn_gscv, param_grid=param_grid_knn, scoring='accuracy', cv=5)
knn_gscv.fit(x_train, y_train)'''
```

Out[ ]:

```
"param_grid_knn = {\n    'n_neighbors': np.arange(5,15),\n    'p': (1,2),\n    'weights': ('uniform', 'distanc
e'),\n    'metric': ('minkowski', 'chebyshev')\n} \nknn_gscv = KNeighborsClassifier()\nknn_gscv = GridSearchCV
(knn_gscv, param_grid=param_grid_knn, scoring='accuracy', cv=5)\nknn_gscv.fit(x_train, y_train)"
```

In [ ]:

```
#print("Best hyperparameters:", knn_gscv.best_params_)
#print("Best accuracy score:", knn_gscv.best_score_)
```

Best hyperparameters: {'metric': 'minkowski', 'n_neighbors': 12, 'p': 2, 'weights': 'uniform'}

Best accuracy score: 0.8124583333333334

In [ ]:

```
knn = KNeighborsClassifier(n_neighbors=12, weights='uniform', p=2, metric='minkowski')
knn.fit(x_train, y_train)
```

Out[ ]:

```
▾      KNeighborsClassifier
KNeighborsClassifier(n_neighbors=12)
```

In [ ]:

```
y_pred_knn = knn.predict(x_test)
```

In [ ]:

```
print("Accuracy : " + str(accuracy_score(y_test, y_pred_knn)))
```

```
Accuracy : 0.8105
```

Support Vector Classification

In [ ]:

```
from sklearn.svm import SVC
```

In [ ]:

```
# commented because it took a whopping 84 minutes to execute!
'''
param_grid_svm = {'C': [0.1, 1, 10, 100, 1000],
                  'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                  'kernel': ['rbf']}

svm_gscv = GridSearchCV(SVC(), param_grid_svm, refit = True, verbose = 3)

# fitting the model for grid search
svm_gscv.fit(x_train, y_train)
'''
```

Out[ ]:
```
"\nparam_grid_svm = {'C': [0.1, 1, 10, 100, 1000],\n\t\t\t'gamma': [1, 0.1, 0.01, 0.001, 0.0001],\n\t\t\t'kern
el': ['rbf']}\n\nsvm_gscv = GridSearchCV(SVC(), param_grid_svm, refit = True, verbose = 3)\n\n# fitting the mo
del for grid search\nsvm_gscv.fit(x_train, y_train)\n"
```

In [ ]:
```
#print("Best hyperparameters:", svm_gscv.best_params_)
#print("Best accuracy score:", svm_gscv.best_score_)
```

Best hyperparameters: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}

Best accuracy score: 0.8206249999999999

In [ ]:
```
sv = SVC(C=1000, gamma= 0.001, kernel='rbf')
sv.fit(x_train, y_train)
```

Out[ ]:
```
▾          SVC
SVC(C=1000, gamma=0.001)
```

In [ ]:
```
y_pred_svc = sv.predict(x_test)
```

In [ ]:
```
print("Accuracy : " + str(accuracy_score(y_test, y_pred_svc)))
```

```
Accuracy : 0.817
```

Logistic Regression Classifier

In [ ]:
```
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
param_grid_log = { 'solver': ['newton-cg', 'lbfgs', 'liblinear'],
                   'penalty': ['l2'],
                   'C' : [100, 10, 1.0, 0.1, 0.01] }

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
log_gscv = GridSearchCV(LogisticRegression(), param_grid_log, n_jobs=-1, cv=cv, scoring='accuracy',error_score=
log_gscv.fit(x_train, y_train)
```

Out[ ]:
```
▸          GridSearchCV
▸ estimator: LogisticRegression
      ▸ LogisticRegression
```

In [ ]:
```
print("Best hyperparameters:", log_gscv.best_params_)
print("Best accuracy score:", log_gscv.best_score_)
```

```
Best hyperparameters: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
Best accuracy score: 0.8120972222222222
```

In [ ]:
```
log = LogisticRegression(C=100, penalty='l2', solver='lbfgs')
log.fit(x_train, y_train)
```

Out[ ]:
```
▾   LogisticRegression
LogisticRegression(C=100)
```

In [ ]:
```
y_pred_log = log.predict(x_test)
```

In [ ]:
```
print("Accuracy : " + str(accuracy_score(y_test, y_pred_log)))
```

```
Accuracy : 0.8053333333333333
```

Bagging Classifier

In [ ]:
```
from sklearn.ensemble import BaggingClassifier
```

In [ ]:
```
'''param_grid_bag = { 'n_estimators': [10, 100, 1000] }

#cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
bag_gscv = GridSearchCV(BaggingClassifier(), param_grid_bag, n_jobs=-1, cv=5, scoring='accuracy',error_score=0)
bag_gscv.fit(x_train, y_train)'''
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: BaggingClassifier
    ▸ BaggingClassifier
```

In [ ]:

```
#print("Best hyperparameters:", bag_gscv.best_params_)
#print("Best accuracy score:", bag_gscv.best_score_)
```

```
Best hyperparameters: {'n_estimators': 1000}
Best accuracy score: 0.8165416666666667
```

Best hyperparameters: {'n_estimators': 1000} Best accuracy score: 0.8165416666666667

In [ ]:

```
bag = BaggingClassifier(n_estimators=1000)
bag.fit(x_train, y_train)
```

Out[ ]:

```
▾ BaggingClassifier
BaggingClassifier(n_estimators=1000)
```

In [ ]:

```
y_pred_bag = bag.predict(x_test)
```

In [ ]:

```
print("Accuracy : " + str(accuracy_score(y_test, y_pred_bag)))
```

```
Accuracy : 0.8143333333333334
```